Automated Evolutionary Program Repair in Expertiza

Zhewei Hu
North Carolina State University
Department of Computer Science
Raleigh, NC 27695-8206
+1 919-579-1902
zhu6@ncsu.edu

ABSTRACT

This proposal describes the research plan for using evolutionary computation to fix runtime exceptions in Expertiza automatically.

Keywords

Automated program repair; evolutionary computation; genetic programming; genetic algorithm

1. PROPOSED WORK

Software maintenance, especially bug fixing is a time-consuming work. Most of the time, software developers need to manually localize the fault, make the change and run the test suite. If any test cases fail, developers have to find the reason, make the changes and run the test again, until all the test cases are passed. In the real world, there are many more bugs than development resources can handle. Former research shows that combining program analysis methods with evolutionary computation can be an effective way to automatically fix program bugs [1]. Researchers developed several tools, such as GenProg [2], ClearView [3], AFix [4], AutoFix-E [5], etc. Different tools focus on different kinds of defects.

In this research, we try to build a pipeline to mitigate the burden on web developers by using genetic programming (GP), a subarea of evolutionary computation [6]. The web application we will use in this research is Expertiza. It is a web application supported by NSF written in Ruby on Rails framework through which students can submit and peer-review learning objects (articles, code, websites, etc). In order to improve the robustness of Expertiza, we use a tool named Airbrake. It is an online tool that provides exception tracking of the web application. [7] When a user triggered an error on our web application, for security reason, we will not display the detailed information to users. Conversely, with the help of Airbrake, developers are able to review errors, tie an error to a certain piece of code and retrieve the backtrace easily, instead of looking into the log file. The bugs reported by Airbrake are runtime exceptions. It is a program error that occurs while the program is running. [8] Comparing to semantic errors, runtime exceptions are more obscure and will cause serious problems during runtime. There are 218 different kinds of exceptions for Expertiza caught by Airbrake in recent 10 months. Some exceptions are triggered more than 1000 times, and others only occur once.

We address two research questions:

RQ1: What percentage of runtime exception errors in Expertiza can be fixed by automated evolutionary program repair via ai4r or darwinning?

RQ2: On average, how long will it take automated evolutionary repair to fix one Expertiza bug?

2. EVALUATION PLAN

Before implementing the pipeline, we have to select a certain number of unresolved runtime exception errors. It is infeasible for us to use all 218 runtime exception errors recorded in Airbrake. The reason is that Expertiza is updating frequently, some errors may no longer exist and some other ones may not be reproducible. Hence we will choose those runtime exception errors which still exist in Expertiza and can be reproducible.

RQ1 evaluation plan:

- Metric: number of runtime exception errors fixed by automated evolutionary program repair out of the total number of selected errors.
- Artifact: unresolved Expertiza runtime exception errors caught by Airbrake.

The first research question answers the effectiveness of newly-designed genetic algorithm (GA). I will try to build a new genetic algorithm for ruby on rails web application based on some existing ruby gems, such as ai4r [9], darwinning [10] and gga4r [11]. Web application runtime exceptions are quite different from other kinds of errors. The input of certain methods can be customized objects with different attributes, instead of predefined objects, such as Integer, String, etc. Therefore, it is not easy for unit tests to decide the effectiveness of GA. So I decide to use GUI test. It can simulate the human activities, such as filling in blanks, submitting forms, etc. One advantage of feature test is that the predefined process has high repeatability and can be used

to check the viability of each variant. Many feature tests already exist in Expertiza. For some other exceptions not covered by the current test suite, I have to create the corresponding feature tests.

RQ2 evaluation plan:

- Metric: average time consumed by automated evolutionary program repair to fix one bug.
- Artifact: unresolved Expertiza runtime exception errors caught by Airbrake.

The second research question answers the efficiency of GA. One of the important steps of the pipeline is fault localization. Only after fault localization can GA start following actions, such as mutate and crossover. Fortunately, Airbrake records the detailed backtrace information of most runtime exception errors. So it is convenient for GA to find the fault locations. However, one bug in the web application may involve several files. As mentioned in Le Goues, et al. [2], comparing with human repair, GA is less likely to find a patch when more files are involved. We need a way to deal with this problem. Maybe we can create several dataflow patterns according to application log file. These patterns will show how data will transfer among different model, view and controller files. During automated bug fixing, GA can look for other related files according to these patterns in order to increase the success rate.

3. TIMELINE

3.1 Unresolved bug selection

- Time period: Oct. 7 to Oct. 13
- Mission: During this time period, I have to select a certain amount of unresolved runtime exception errors recorded in Airbrake. The selection criterion is that selected errors are reproducible and have enough information for GA to use, such as the URL of bug occurrence, backtrace information, etc. Ideally, I hope I can find 50 or more runtime exception errors meet the requirements

3.2 Genetic algorithm implementation

- Time period: Oct. 14 to Oct. 31
- Mission: In this stage, I have to implement GA. According to the available URL, it should be easy to find the fault location. Then it should build the abstract syntax tree (AST) for certain block of code. There are several existing tools to parse ruby code, such as ast [12], parsetree [13], etc. According to the result of AST, I am able to implement genetic operators, such as mutate and crossover. For each variant generated by GA, the next step in the pipeline is to run feature tests written by RSpec [14], capybara [15] and selenium [16] to check the viability. The GA will run iteratively. And I will set several scenarios to terminate this process, such as the number of generations, final variant occurrence, etc.

3.3 Automated bug fixing

- Time period: Nov. 1 to Nov. 17
- Mission: Implementing the pipeline to unresolved Expertiza runtime exception errors. Inspired by Le Goues, et al. [2], I think I will take the advantage of Amazon Web Service to run the code in the cloud and record all the detail for further analysis. If possible, I will adopt parallelism to shorten the operation time.

3.4 Result analysis

- Time period: Nov. 18 to Nov. 23
- Mission: Analyse the effectiveness and the efficiency of the pipeline, especially for GA. The effectiveness of the algorithm can be represented by the percentage of runtime exception errors fixed automatically, and the efficiency of the algorithm can refer to the average bug fixing time. I will also look into those errors not being fixed automatically and try to find the reason under the hood.

3.5 Final paper writing

- Time period: Nov. 24 to Nov. 31
- Mission: Write final paper.

4. REFERENCES

[1] W. Weimer, S. Forrest, C. Le Goues and T. Nguyen, "Automatic program repair with evolutionary computation", Communications of the ACM, vol. 53, no. 5, p. 109, 2010.

[2] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, “A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each,” in ICSE ’12.

[3] J. H. Perkins, S. Kim, S. Larsen, S. Amarasinghe, J. Bachrach, M. Carbin, C. Pacheco, F. Sherwood, S. Sidiroglou, G. Sullivan, W.-F. Wong, Y. Zibin, M. D. Ernst, and M. Rinard, “Automatically patching errors in deployed software,” in Symposium on Operating Systems Principles, 2009.

[4] G. Jin, L. Song, W. Zhang, S. Lu, and B. Liblit, “Automated atomicity-violation fixing,” in Programming Language Design and Implementation, 2011.

[5] Y. Wei, Y. Pei, C. A. Furia, L. S. Silva, S. Buchholz, B. Meyer, and A. Zeller, “Automated fixing of programs with contracts,”

in International Symposium on Software Testing and Analysis, 2010, pp. 61–72.

[6] "Evolutionary computation", Wikipedia, 2016. [Online]. Available:

https://en.wikipedia.org/wiki/Evolutionary_computation. [Accessed: 06- Oct- 2016].

[7] "airbrake/airbrake", GitHub, 2016. [Online]. Available: https://github.com/airbrake/airbrake. [Accessed: 06- Oct- 2016].

[8] "Runtime Error Definition", Techterms.com, 2016. [Online]. Available: http://techterms.com/definition/runtime_error. [Accessed: 06- Oct- 2016].

[9] "SergioFierens/ai4r", GitHub, 2016. [Online]. Available: https://github.com/SergioFierens/ai4r. [Accessed: 07- Oct- 2016].

[10] "dorkrawk/darwinning", GitHub, 2016. [Online]. Available: https://github.com/dorkrawk/darwinning. [Accessed: 07- Oct- 2016].

[11] "spejman/gga4r", GitHub, 2016. [Online]. Available: https://github.com/spejman/gga4r. [Accessed: 07- Oct- 2016].

[12] "whitequark/ast", GitHub, 2016. [Online]. Available: https://github.com/whitequark/ast. [Accessed: 06- Oct- 2016].

[13] "seattlerb/parsetree", GitHub, 2016. [Online]. Available: https://github.com/seattlerb/parsetree. [Accessed: 06- Oct- 2016].

[14] "RSpec: Behaviour Driven Development for Ruby", Rspec.info, 2016. [Online]. Available: http://rspec.info/. [Accessed: 06- Oct- 2016].

[15] "jnicklas/capybara", GitHub, 2016. [Online]. Available: https://github.com/jnicklas/capybara. [Accessed: 06- Oct- 2016].

[16] "paytonrules/selenium-on-rails", GitHub, 2016. [Online]. Available: https://github.com/paytonrules/selenium-on-rails. [Accessed: 06- Oct- 2016].